UNIVERSIDADE ESTADUAL DE CAMPINAS
Instituto de Matemática Estatística e Computação Científica

Novas aplicações em otimização de média sobre grafos

Relatório de renovação para o Programa de Iniciação Científica e Mestrado (PICME)

ALUNO: Kauan Cunha da Silva
ORIENTADOR: Eduardo Garibaldi
COORIENTADOR: João Tiago Assunção Gomes
VIGÊNCIA: Fevereiro de 2025 a Junho de 2025

Resumo

No primeiro semestre de 2025, exploramos a Álgebra Min-Plus, também conhecida como Álgebra Tropical, com o objetivo de ampliar o repertório de ferramentas para o estudo da Otimização de Média sobre Grafos. Este relatório tem a finalidade de documentar e resumir os conteúdos abordados ao longo do período, bem como apresentar as conclusões obtidas a partir da comparação entre as ferramentas tropicais e os algoritmos estudados no semestre anterior.

Sumário

1 INTRODUÇÃO

O artigo aborda o problema de projetar linhas de trem e ônibus de forma eficiente. Para isso, ele apresenta uma maneira de analisar a tabela de horários de uma linha de trem, representando-a como um Sistema Dinâmico de Eventos Discretos (DEDS). Além disso, o problema é relacionado à busca do Conjunto Cíclico Maximal. Nesse contexto, o artigo estuda a implementação do algoritmo clássico de Tarjan, em conjunto com o algoritmo de Floria-Griffiths.

2 SISTEMA DINÂMICO DE EVENTOS DISCRETOS (DEDS)

Um DEDS pode ser descrito da seguinte forma [REF5]:

- $\mathcal{S}$:= Conjunto dos Eventos (Saídas de trem/ônibus).
- $\mathcal{R}$:= Conjunto do recurso necessário para que haja evento (Carro de Trem ou Ônibus).
- $\mathcal{U}$:= Conjunto das relações entre elementos de $\mathcal{S}$, que define uma ordem.
- P := Conjunto das sequencias de eventos.

Graficamente, um DEDS pode ser representado como um grafo, no qual arcos e vértices recebem pesos:

- δ_{ij} := (Tempo para trocar da atividade/evento i para o j)
- t_{ij} := (Tempo para se realizar evento s_i) + δ_{ij}

No problema discutido no artigo, esses pesos correspondem aos horários de partida e ao intervalo entre elas.

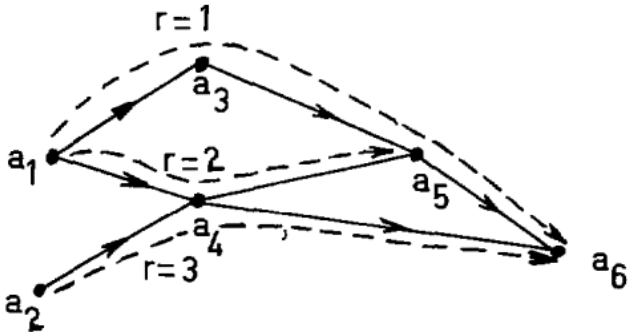

Figura 1 – Exemplo de sistema de eventos discretos

Nesse contexto, podemos definir o conceito de "tempo possível mais cedo"(x_i) para que o evento s_i ocorra. Se $(j, i) \in \mathcal{U}$, então $x_i \geq x_j + t_{ji}$. Isto é, x_i obviamente é maior do que a soma do tempo inicial da última atividade mais o tempo de sua realização, pois ambas compartilham o mesmo recurso. Assim, temos:

$$\forall s_i \in \mathcal{S}, \quad x_i = max(x_j + t_{ji})$$

3 GRAFO PRECEDENTE

No contexto do problema, o sistemas de linha de trens erá representado pelo grafo precedente, i.e, uma sistema dinâmico de eventos discretos. Esse grafo descreve todas as restrições prévias do sistema. Na fig. 2 temos a representação de uma linha férrea simples, com duas estações e suas restrições. Na figura 3 temos que os nos representam as partidas de trens da estação, e os pesos s_{ji} associado (i, j) será a soma t_i^r,tempo de parada/transferência, e o tempo de deslocamento de uma partida i a j.

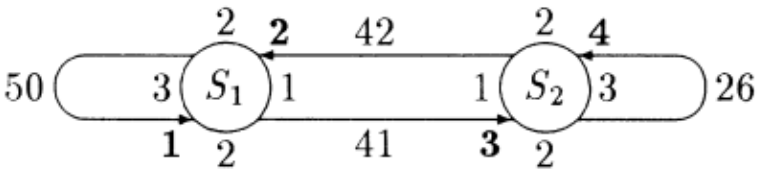

Figura 2 – Sistema Férreo

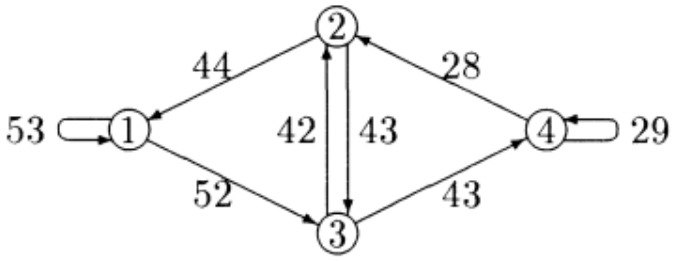

Figura 3 – Grafo precedente da fig. 2

Note que a média do pesos de um ciclo no grafo precedente é o tempo médio de viagem nele. Ou seja, o tempo médio de viagem no ciclo $S_1S_2S_1$ é dado por $\frac{42+43}{2}$.

Uma vez que estamos no contexto de Sistema Dinâmicos de Eventos Discretos, definimos o "tempo possível de partida mais cedo", x_i. Considerando que o sistema é começa de uma partida unitária inicial:

$$x_i := \max(s_{ij} + x_j), \quad i = 1, ..., n$$

Note que estamos interessados na periodicidade do sistema. Assim, chamando $x_i(k)$ de k-ésimo tempo de partida do trem i, temos a periodicidade implantada da seguinte forma:

$$x(k+1) = \max(a_{ij} + x_j(k)), \quad i = 1, ..., n$$

Uma vez que estamos interessados em analisar linhas reais, podemos ainda considerar d_i, o tempo de partida agendado:

$$d_i(k) = d(0) + kT, \quad T := \text{período desejado}$$

$$x_i = \max(\max(a_{ij} + x_j(k)), d_i(k+1))$$

Considerar o tempo do cronograma é interessante para gerar uma variável de controle, r_{ji}:

$$r_{ji}(k+1) = d_i(k+1) - (a_{ij} + x_j(k))$$

Se o valor é pequeno, o sistema é sensível a atrasos, i.e, um atraso de 10 minutos na parte da manhã gera uma cadeia de atrasos durante o dia inteiro. Porém, um valor muito grande gera um sistema com tempo de espera indesejado. Estudar essa variável de controle é importante para sistemas seguros.

4 TARJAN

Em teoria de grafos, uma **componente fortemente conexa (CFC)** é um subconjunto de vértices de um grafo direcionado no qual qualquer vértice pode ser alcançado a partir de qualquer outro dentro do mesmo conjunto. Em outras palavras, se dois vértices estão na mesma componente fortemente conexa, existe sempre um caminho de ida e volta entre eles.

O algoritmo de Tarjan é um método eficiente para encontrar todas as componentes fortemente conexas de um grafo direcionado. Ele percorre o grafo usando uma busca em profundidade (DFS), mantendo uma pilha de vértices visitados e calculando dois valores importantes para cada nó:

- o *índice de descoberta*, que indica a ordem em que o vértice foi visitado;
- o *menor índice alcançável* a partir dele (considerando seus vizinhos).

Uma vez que o DFS de um vértice retorna e o seu menor índice alcançável é ele mesmo, ele é a "raiz"da componente e desempilhamos a pilha contendo todos seus descendentes definindo assim, uma componente.

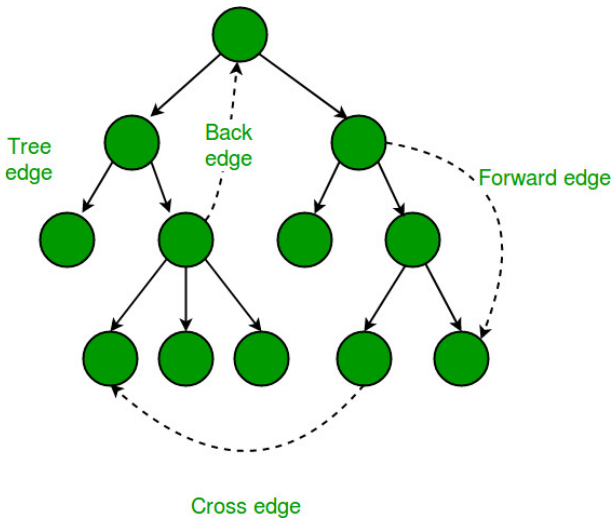

Figura 4 – Árvore de uma DFS

Na figura 4 temos os dois tipos de arestas que nos importa, perceba que a aresta de retorno garante que a subárvore de onde ela parte até onde ela chega é conexo. Além disso,

"cross"nós pode quebra a lógica do algoritmo caso acesse um ramo já visitado, portanto, só percorremo-os se apontar para um nó não visitado (ps. por isso armazenamos os nós visitados na implementação). A seguir uma ilustração do algoritmo em ação:

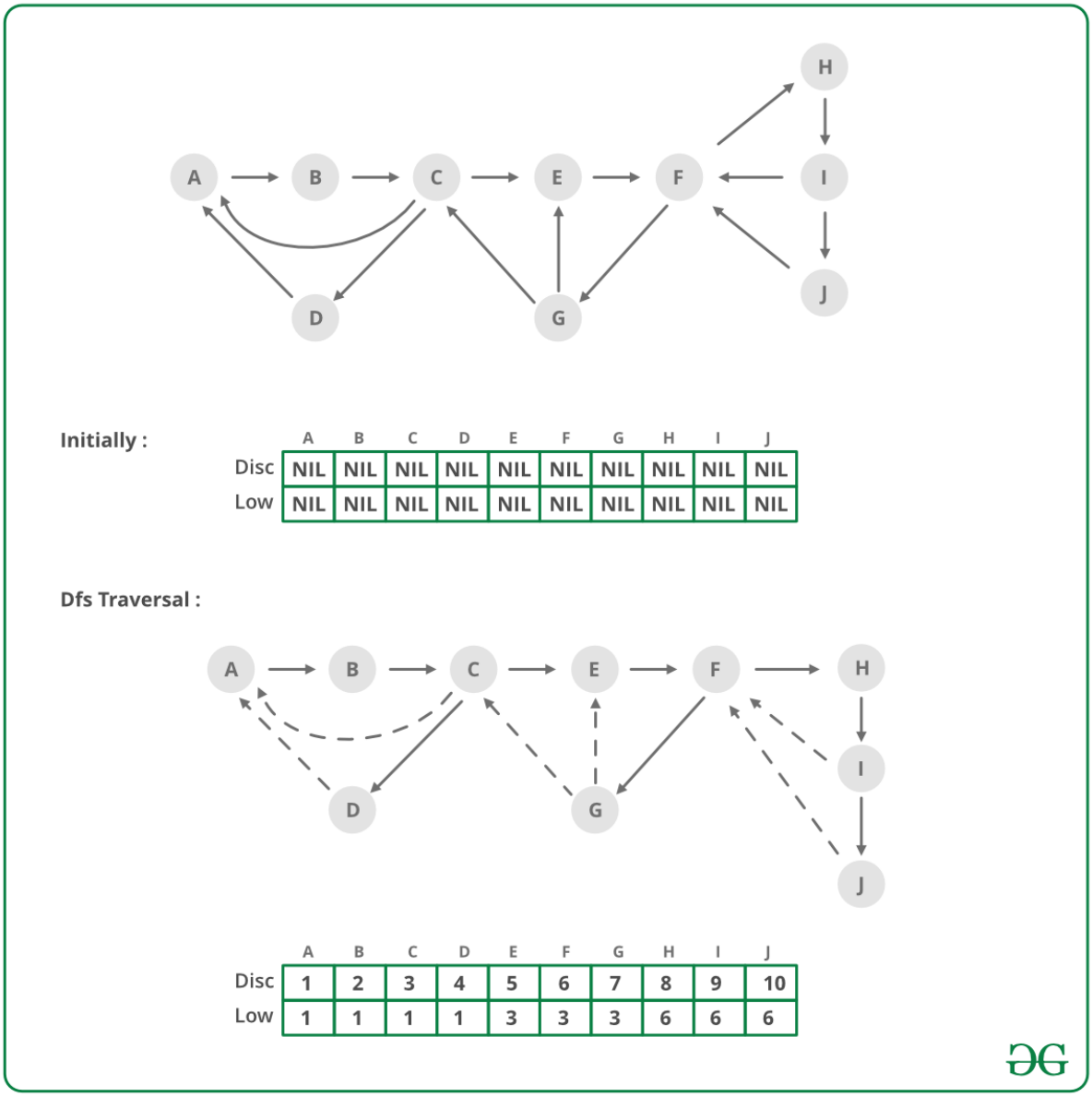

Figura 5 – Tarjan

```
def tarjan_rec(adj:list, raiz: int, disc:list, low:list, visited:list,
    pilha: list, naPilha:list, current_disc):
    pilha.append(raiz)
    naPilha[raiz] = True
    visited[raiz] = True
    disc[raiz] = current_disc[0]
    low[raiz] = disc[raiz]
    current_disc[0] += 1

    for i in range(len(adj)):
        if adj[raiz][i] == 1:
```

```
            if naPilha[i] == True:        #caso I: aresta de retorno
                low[raiz] = min(low[raiz], disc[i])

            elif visited[i] == False:       #caso II: aresta descendente
                tarjan_rec(adj, i, disc, low, visited, pilha, naPilha,
    current_disc)
                low[raiz] = min(low[raiz], low[i])

    # Se ao retornar o low e o disc forem iguais, a raiz e "cabeca" da SSC
    if low[raiz] == disc[raiz]:
        while pilha[-1] != raiz:
            retirado = pilha.pop()
            low[retirado] = low[raiz]
            naPilha[retirado] = False

        naPilha[pilha.pop()] = False

def tarjan(adj):
    pilha = []
    naPilha = [False] * len(adj)
    current_disc = [0]
    visited = [False] * len(adj)
    disc = [-1] * len(adj)
    low = [-1] * len(adj)

    for i in range(len(adj)):
        if visited[i] == False:
            tarjan_rec(adj, i, disc, low, visited, pilha, naPilha,
    current_disc)

    return low


# Exemplo de como usar:
grafo_adj = [
    [ 0, 1, 0, 0, 0 ],
    [ 1, 0, 0, 0, 0 ],
    [ 0, 0, 0, 1, 0 ],
    [ 0, 0, 0, 0, 1 ],
    [ 0, 0, 1, 0, 0 ]
]

componentes = tarjan(grafo_adj)
print(componentes)
# Retorno: [0, 0, 2, 2, 2]
```

Listing 1 – Implementação do Algoritmo de Tarjan para encontrar componentes fortemente conexos.

Essa foi a implementação que cheguei lendo a descrição do algoritmo, por tanto, existe

algumas otimizações que podem (e DEVEM) ser feitas, as mais significativas são: aceitar a representação de grafos em vetor de lista ligada (só assim é possível percorre-lo em O(log(V+E)), fazer a checagem de visita utilizando o vetor descoberta (se não visitado, ent disc[i] = -1).

REFERÊNCIAS

FLORIA, Luis M; GRIFFITHS, Robert B. Numerical procedure for solving a minimization eigenvalue problem. **Numerische Mathematik**, 1989. Disponível em: <https://doi.org/10.1007/BF01398916>.

GARIBALDI, Eduardo; ASSUNçãO, João Tiago. **Curso Introdutório: Otimização de Médias sobre Grafos Orientados**. 2013. Disponível em: https://www.ime.unicamp.br/ garibaldi/-minicurso/. Acesso em: janeiro 2025.

GOVERDE, Rob. **The max-plus algebra approach to railway timetable design**. 1998. Disponível em:https://www.witpress.com/Secure/elibrary/papers/CR98/CR98033FU.pdf. Acesso em: Junho 2025.